Computer Networks X_400487

Lecture 3

Chapter 3: The Data Link Layer—Part 2

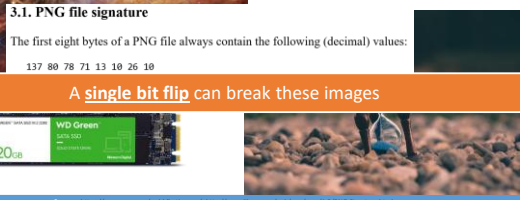
Lecturer: Jesse Donkervliet

Data Link Layer — Roadmap

Part 1
- Framing
- Flow Control
- Guaranteed Delivery
- Sliding Window Protocols

Part 2
- **Error detection**
- **Error correction**

1 Gibibyte = 8×2^{30} bits



A **single bit flip** can break these images

Sources: https://www.nasa.gov/webfiles-images/ ; http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html ; https://pngcheck.com/photos/PCKVOV18.7z/

Error Detection

Detecting errors in received frames

Q: What causes these bit flips?

Data at sender: 01110101010101110101 0001

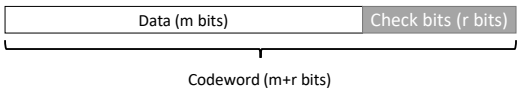
Data at receiver: 01110101**1**1010111010 0001

Somehow bit was flipped!

Adding redundant bits

For a message of m bits,
send an extra r redundant bits.

Send $m + r$ to the receiver. ← Systematic code



Hamming distance

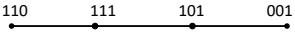
Number of bits that differ between two bit-strings

10001001
10110001

Three bit flips required to change from one sequence to the other.

Adding redundant bits increases the distance between valid bit strings!

Hamming distance: 3

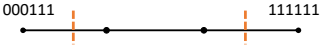


How many errors can we detect?

Consider code words:
000111
111111
000000

Q: How many single-bit errors can we detect?

Hamming distance 3,
so we can detect $3 - 1 = 2$ single-bit errors.



Error detection

Q: How to assess the quality of these codes?

Linear, systematic block codes:
1. Parity
2. Checksums
3. Cyclic Redundancy Checks (CRCs)

Block is $n = m + r$ bits large
Important code properties:
1. Code Ratio: $\frac{m}{n}$
2. How many / which kind of errors are reliably detected?

Parity

Q: How many bit errors can be detected?

Add single bit such that:
- The sum of the data bits modulo 2 is 0
- The number of 1's is even

Send example: 1110000 → 1110000**1**
Receive example: 11010101 ← Error detected!

Easy to detect an *odd* number of errors

Parity

Q: How many bit errors can be detected?

Add single bit such that:
- The sum of the data bits modulo 2 is 1
- The number of 1's is odd

Send example: 1110000 → 1110000**0**
Receive example: 11010100 ← Error detected!

Easy to detect an *odd* number of errors

Multiple parity bits

transmit order →

11100000010101110101001010001111000

Multiple parity bits

transmit order →

1110000 → 1
0010101 → 1
1101010 → 0
0101000 → 0
1111000 → 0

Multiple single-bit errors detected.

14

Burst errors

transmit order →

1110000 → 1
0010101 → 1
1101010 → 0
0101000 → 0
1111000 → 0

channel →

Burst error not detected!

0011100 → 1
0010101 → 1
1101010 → 0
0101000 → 0
1111000 → 0

15

Burst errors

transmit order →

1110000
0010101
1101010
0101000
1111000
↓↓↓↓↓↓↓
1011111

channel →

Burst error detected.

0011100
0010101
1101010
0101000
1111000
↓↓↓↓↓↓↓
1011111

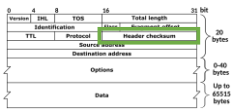
16

Checksums

Q: Disadvantages?

Checksum treats data as N-bit words and adds N check bits that are the modulo 2^N sum of the words.

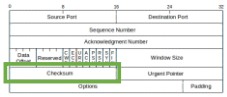
Example: Internet (i.e., IPv4) 16-bit one's complement header checksum



Used in IPv4 (network layer)

Properties:

- Improved error detection over parity bits.
- Detects bursts up to N errors.
- Vulnerable to systematic errors, e.g., added zeros.



Used in TCP (transport layer)

17

Checksum Example

Groups of 8 bits → calculate *sum mod 256*

transmit order →

11101000
00100101
11011010
01010000
11111000

add

00000000
11101000
---------- +
11101000

Internet checksum uses one's complement arithmetic

18

Checksum Example

Groups of 8 bits → calculate *sum mod 256*

transmit order →

11101000
00100101
11011010
01010000
11111000

add

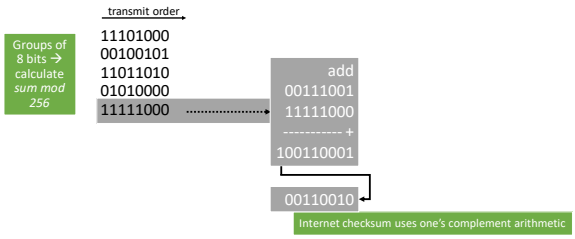
11101000
00100101
---------- +
100001101

00001110 ← (carry added back: one's complement arithmetic)

Internet checksum uses one's complement arithmetic

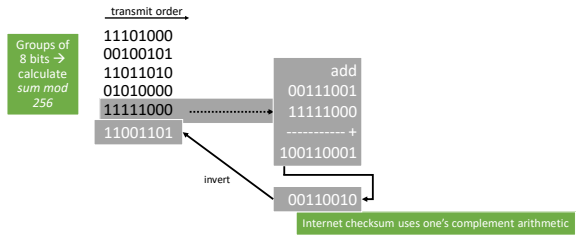
19

Checksum Example



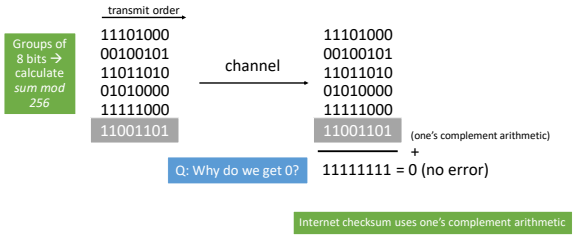
20

Checksum Example



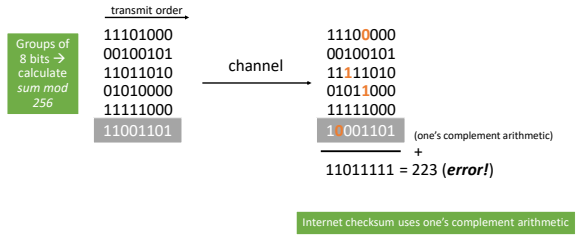
21

Checksum Example



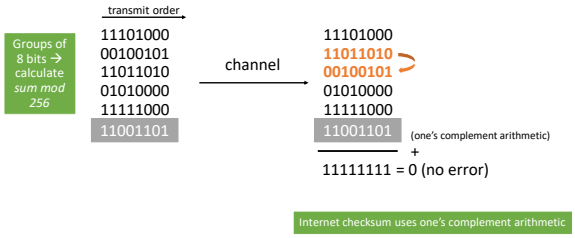
22

Checksum Example



23

Checksum Example



24

Checksum Example #2

To send: AB A8 EF F3

Compute 8-bit (1 byte) word checksum:

```
 AB                          B(11)+8+F(15)+3=37
 A8                          37 = 2×16 + 5
 EF                             = 0x25 (carry = 2)
 F3
 ——+ (1's complement)        2+A(10)+A+E(14)+F=51
??5                          51 = 3×16 + 3
335                             = 0x33
```

26

Checksum Example #2

To send: AB A8 EF F3
Compute 8-bit (1 byte) word checksum:

```
AB                              B(11)+8+F(15)+3=37
A8                              37 = 2×16 + 5
EF                                 = 0x25 (carry = 2)
F3
—+ (1's complement)             2+A(10)+A+E(14)+F=51
38  (335 → 3 + 35 = 38)         51 = 3×16 + 3
                                   = 0x33
```


Checksum Example #2

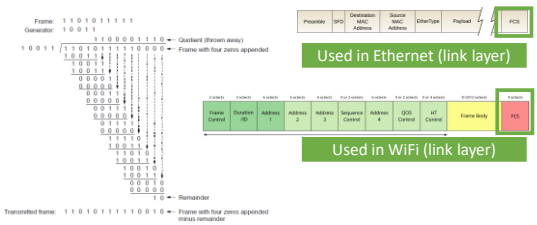
To send: AB A8 EF F3
Compute 8-bit (1 byte) word checksum:

```
AB
A8
EF
F3
—+ (1's complement)
38  (335 → 3 + 35 = 38)
```

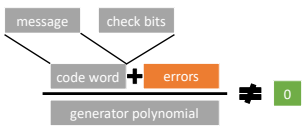
```
Checksum is inverse
of 38:

3 → C (15-3=12=C)
8 → 7 (15-8= 7=7)

Checksum: C7
```


Lecture resumes at 14:18

Cyclic Redundancy Check



Cyclic Redundancy Check
The concept



Cyclic Redundancy Check
Properties and practice

Sender and receiver agree upon polynomial in advance
Example: Ethernet's 33-bit polynomial is:
$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$
CRC is computed with simple shift/XOR circuits
Because we use modulo 2 arithmetic.
Stronger detection than checksums:

1. Can detect all double bit errors, odd bit errors
2. Detect all burst errors $\leq r$ bits (in example, $r = 32$)
3. Not vulnerable to systematic errors
4. …

Cyclic Redundancy Check

Cyclic Redundancy Check Example

Sender adds CRC

$\mathbf{1} \times x^4 + \mathbf{0} \times x^3 + \mathbf{0} \times x^2 + \mathbf{1} \times x^1 + \mathbf{1} \times x^0$

message: 11010101**0000**
generator: 10011

$x^4 + x + 1$

```
10011010000
10011
   10011
   10011
    0011
```

Modulo 2 arithmetic. No carries/borrows

Q: Consequences for implementation?

Message: 11010101, CRC: 0011,
Codeword: 11010101**0011**

$\frac{11010101 0011}{10011} = 0$

Cyclic Redundancy Check Example

Receiver checks for errors

message: 11010101**0011**
generator: 10011

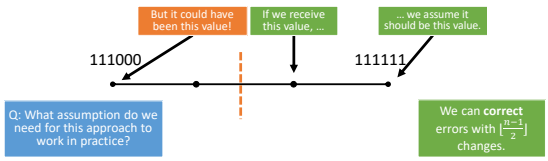
```
10011010011
10011
   10011
   10011
       0
```

$\frac{11010101 0011}{10011} = 0$, no error

Error Correction

How many errors can we *correct*?

Consider a code with hamming distance n.
We have seen that we can **detect** $n - 1$ single-bit errors.



Error correction

1. Hamming codes
2. Binary convolutional codes
3. Reed-Solomon codes
4. Low-Density Parity Check codes
5. … (many others)

Multiple parity bits

receive order →

1110000 → 1
0010101 → **0**
1101010 → 0
0101000 → 0
1111000 → 0

Error in second row!

Multiple parity bits

receive order →

1110000
0010101
1101010
0101000
1111000
↓↓↓↓↓↓↓
1010111

Error in fourth column!

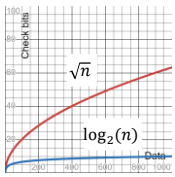
Multiple parity bits

receive order →

1110000 → 1
0010101 → **0**
1101010 → 0
0101000 → 0
1111000 → 0
↓↓↓↓↓↓↓
1010111

Error located!

= Row+column parity bits
= Hamming code



Hamming codes

Minimum number of check bits such that *all* 1-bit errors can be *corrected*!

Example of an (11, 7) Hamming code correcting a single-bit error.

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits. Use the remaining positions for the message.

message: 11010101
codeword: __1_101_0101
positions: 123456789…

Hamming codes
An example

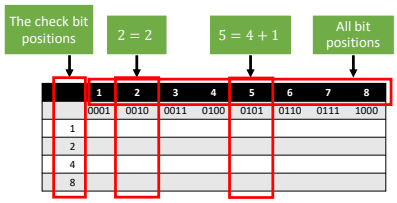
Use bit-locations that are a power of 2 as check bits. Use the remaining positions for the message.

message: 1 101 0101
codeword: __1_101_0101
positions: 123456789…

1. Expand all bit locations into powers of two.
2. Decide the value of each check bit in position 2^i by calculating the parity function over all bits that have 2^i in their expansion.

Hamming codes
An example

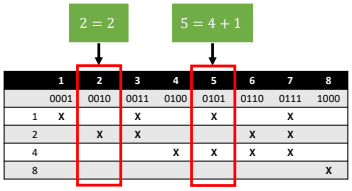
1. Expand all bit locations into powers of two.



45

Hamming codes
An example

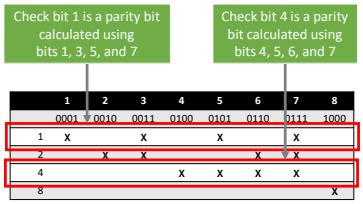
1. Expand all bit locations into powers of two.



46

Hamming codes
An example

2. Calculate the parity bit using all bits that have 2^i in their expansion



47

Hamming codes
An example

2. Calculate the parity bit using all bits that have 2^i in their expansion

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101
codeword: __1_101_0101
positions: 123456789…

Position 1+2
Position 4
Position 4+2+1

48

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits. Use the remaining positions for the message.

message: 1 101 0101
codeword: __1_101_0101
positions: 123456789…

49

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits. Use the remaining positions for the message.

message: 1 101 0101
codeword: 1_1_101_0101
positions: 123456789…

50

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits. Use the remaining positions for the message.

message: 1 101 0101

codeword: 1_**1**_101_**0**1**0**1

positions: 1**2**3456789…

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits. Use the remaining positions for the message.

message: 1 101 0101

codeword: 1**1**1_101_**0**1**0**1

positions: 1**2**3456789…

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits. Use the remaining positions for the message.

message: 1 101 0101

codeword: 111110100101

positions: 123456789…

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101 ← Single-bit error

codeword: 111111100101

positions: 123456789…

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101 ← Single-bit error

codeword: 1_1_1_1**1**00_101

positions: 123456789…

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11 = 0

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101 ← Single-bit error

codeword: 1_1_1_1**1**00_101

positions: 123456789…

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11 = 0

Check bit 2 = parity of bits 2, 3, 6, 7, 10, 11 = 1

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101
codeword: 111111100101
positions: 123456789…

Single-bit error
Error at location: 110 (binary)

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11 = 0
Check bit 2 = parity of bits 2, 3, 6, 7, 10, 11 = 1
Check bit 4 = parity of bits 4, 5, 6, 7, 12 = 1

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101
codeword: 111111100101
positions: 123456789…

Single-bit error
Error at location: 110 (binary)

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11 = 0
Check bit 2 = parity of bits 2, 3, 6, 7, 10, 11 = 1
Check bit 4 = parity of bits 4, 5, 6, 7, 12 = 1

Computer Networks
X_400487

Lecture 3
Chapter 3: The Data Link Layer—Part 2

Lecturer: Jesse Donkervliet

VU Vrije Universiteit Amsterdam

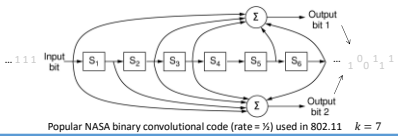
Extra Slides

Convolutional codes

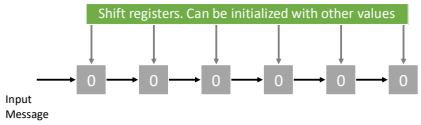
Different from *systematic* codes and *block* codes

Operates on a stream of bits, keeping internal state.
Output stream is a function of last k preceding input bits.
Bits are decoded with the Viterbi algorithm.

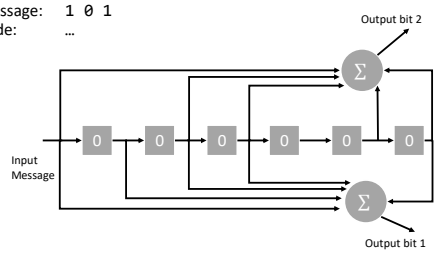
Determines most likely input for given output.



Popular NASA binary convolutional code (rate = ½) used in 802.11 $k = 7$



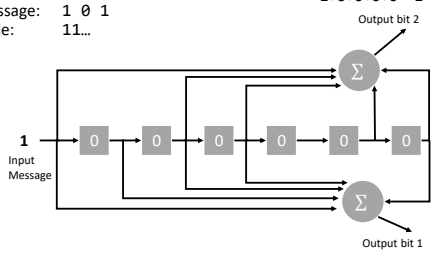
Message: 1 0 1
Code: …



63

Message: 1 0 1
Code: 11…

1+0+0+0+0 = 1

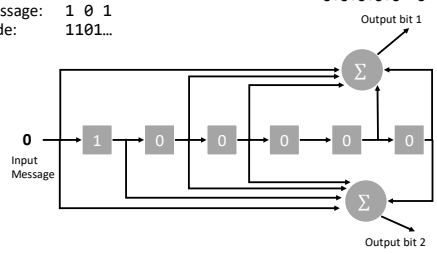


1+0+0+0+0 = 1

64

Message: 1 0 1
Code: 1101…

0+0+0+0+0 = 0

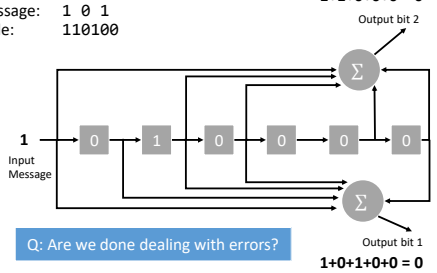


0+1+0+0+0 = 1

65

Message: 1 0 1
Code: 110100

1+1+0+0+0 = 0



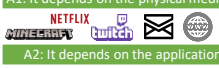
Q: Are we done dealing with errors?

1+0+1+0+0 = 0

66

Data Link Layer Summary

A1: It depends on the physical medium



A2: It depends on the application

Framing (byte stuffing, bit stuffing, etc)
Guaranteed delivery Q: When needed?
• Sequence numbers, acknowledgments, and retransmissions
Flow control Q: When needed?
• Stop-and-Wait
• Sliding Window
Error control Q: When needed?
• Detection (e.g., Parity bit, Checksum, CRC, …)
• Correction (e.g., Hamming Code, Convolutional Code, …)

A3: We may want to address the problem in a higher layer
Q: Can you think of an example?

67